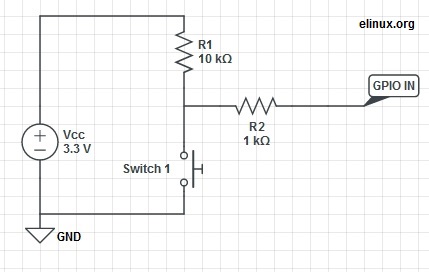
Buttons and switches are a fundamental part of ‘physical’ computing. Your basic switch is usually required to control the on/off status of a circuit. However with a little programming from your Raspberry Pi, that same switch can be transformed into an incredibly versatile tool! You’ll be able to control I/O, start, stop, skip or run through a program (like telling a robot to go on a mission), set a system status by holding a switch for a period of time, or complete a multitude of other tasks that a switch alone cannot accomplish! Whether it is a switch, a sensor or any other physical component – being able to control and interact via programming with signals from these parts, such as the push of a button, is the basis of many physical computing projects. This beginner’s tutorial is designed to teach the basics of physical operation and programming with the Raspberry Pi using a simple momentary switch setup.

The tutorial requires a few simple components which are available from ModMyPi (product codes in brackets):

* Medium Breadboard (BB2) – For laying out our components & circuit.
* Male to Female Jumper Wires (JW8) – For jumping between the RPi & breadboard.
* PCB Mount Switch (TAC001) – A four point basic momentary switch.
* ModMyPi’s Ridiculous Resistor Kit (RK995) – To protect our Pi & calibrate the float voltage.
  + 10KΩ Resistor - (Brown, Black, Black, Red, Brown)
  + 1KΩ Resistor - (Brown, Black, Black, Brown, Brown)
* Breadboard Jumper Wire Kit (140KI) – For easy jumping on the breadboard.

**The Circuit**

The purpose of this circuit is to enable the Raspberry Pi to detect a change in voltage and run a program when the button (***Switch 1***) is pressed. This requires three GPIO pins on our Raspberry Pi: the first will provide a signal voltage of 3.3V (***Vcc***), the next will ground the circuit (***GND***), and the third will be configured as an input (***GPIO IN***) to detect the voltage change.

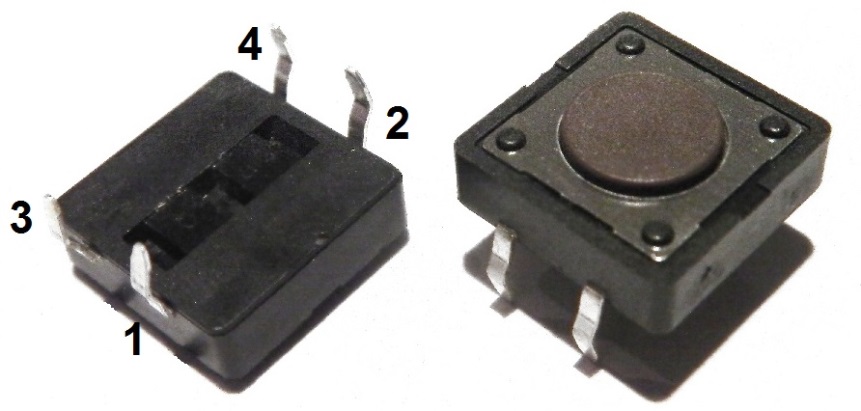
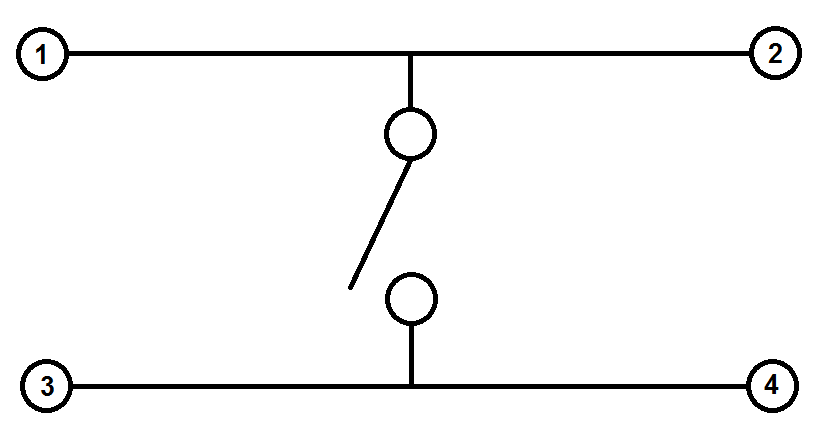
When a GPIO pin is set to input, it doesn’t provide any power and consequently has no distinct voltage level; defined as ‘floating’. We need the pin to be capable of judging the difference between a high and low voltage, however in a floating state it’s liable to incorrectly detect states due to electrical noise. To enable the pin to see the difference between a high or low signal we must ‘tie’ that pin, calibrating it to a defined value; 3.3V in this case!

To tie the input pin, we connect it to the ***Vcc*** 3.3V pin, hence when ***Switch 1*** is open, the current flows through ***GPIO IN*** and reads high. When ***Switch 1*** is closed, we short the circuit and the current is pulled to ***GND***; the input has 0V, and reads low! The large ***R1*** (10kΩ) resistor in this circuit ensures that only a little current is drawn when the switch is pressed. If we don’t use this resistor, we are essentially connecting ***Vcc*** directly to ***GND***, which would allow a large current to flow, potentially damaging the Pi! To make the circuit even safer in case we get something wrong, we add the ***R2*** (1kΩ) resistor to limit the current to and from ***GPIO IN***.

**The Switch**

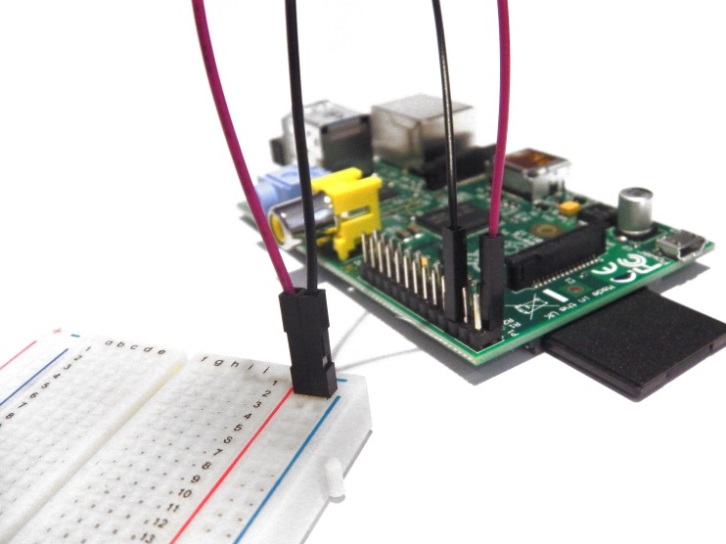
Four point switches are wired in a very similar manner to two point switches. They’re simply more versatile, as you can have multiple isolated inputs into the same switching point. Checking the diagrams, Pins 1 & 2 are always connected, as are Pins 3 & 4. However, both pin pairs are disconnected from each other when the button is not pressed e.g. Pins 1 & 2 are isolated from Pins 3 & 4. When the button is pressed the two sides of the switch are linked and Pins 1, 2, 3 & 4 are all connected!

In ‘momentary’ switches the circuit disconnects when pressure is removed from the button, as opposed to ‘toggle’ switches when one push connects and the next push disconnects the internal switch circuit.

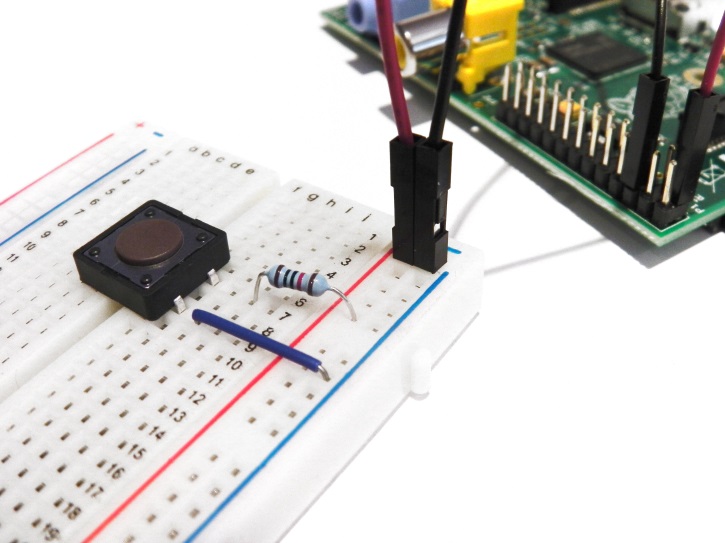


**Where does it all go?**

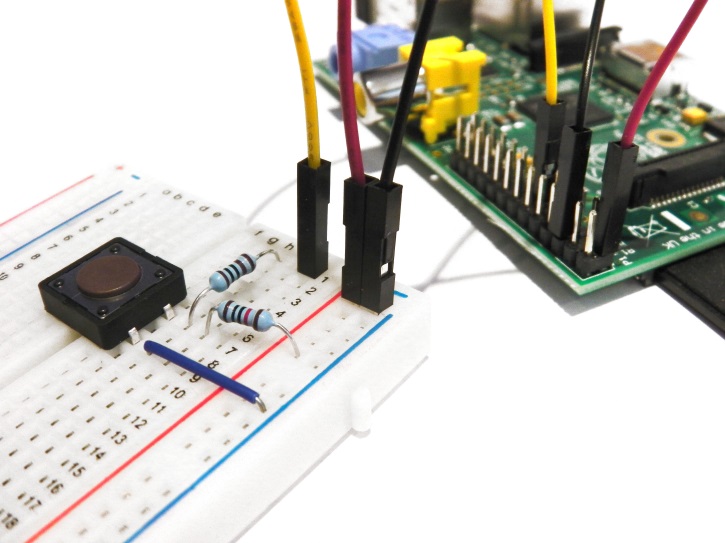
**WARNING.** When hooking up to GPIO points on your Raspberry Pi care must be taken, as connecting the wrong points could permanently fry your Pi. Please use a GPIO cheat-sheet, and double check everything before switching it on. I will denote each GPIO point by its name, and physical location, for example GPIO P17 is actually located at Pin 11, denoted: ***GPIO P17 [Pin 11]***. The irregularities are a result of the pin names being referenced by the on board chip rather than their physical location.

****1. Connect Pi to Ground Rail**. Use a black jumper wire to connect ***GPIO GND [Pin 6]*** on the Pi to the Negative rail on the breadboard – the rail on the edge of the board with the negative sign (-).

**2. Connect Pi 3.3V to Positive Rail.** Use a red jumper wire to connect ***GPIO 3.3V [Pin 1]*** on the Pi to the Positive rail on the breadboard – the edge rail with the positive sign (+).

**3. Plug your switch in**. When breadboarding, make sure all of the legs are in separate rows. To achieve this straddle the central channel on the breadboard.

**4. Add 10kΩ Resistor.** Connect this from ***Switch Pin 1***, to the positive (+) rail of the breadboard. Orientation of standard film resistors is unimportant.

****5. Connect Switch to Ground.** Use a breadboard jumper wire to hook ***Switch Pin 3*** to the ground (-) rail.

**6. Connect Switch to 1kΩ Resistor.** Add this resistor between ***Switch Pin 1*** and the ***10kΩ Resistor*** and take it to a clear rail.

**7. Connect Switch to Signal Port.** We’ll be using ***GPIO P17*** to detect the 3.3V signal when the switch is pressed. Simply hook up a jumper between ***GPIO P17 [Pin 11]*** on the Pi and the ***1kΩ Resistor*** rail.

**That’s our circuit built! Next time, we’ll write a simple program in Python to run when we press the switch!**